Video Visualization

Gareth Daniel Min Chen
University of Wales Swansea, UK [*]

Abstract

Video data, generated by the entertainment industry, security and traffic cameras, video conferencing systems, video emails, and so on, is perhaps most time-consuming to process by human beings. In this paper, we present a novel methodology for "summarizing" video sequences using volume visualization techniques. We outline a system pipeline for capturing videos, extracting features, volume rendering video and feature data, and creating video visualization. We discuss a collection of image comparison metrics, including the linear dependence detector, for constructing "relative" and "absolute" difference volumes that represent the magnitude of variation between video frames. We describe the use of a few volume visualization techniques, including volume scene graphs and spatial transfer functions, for creating video visualization. In particular, we present a stream-based technique for processing and directly rendering video data in real time. With the aid of several examples, we demonstrate the effectiveness of using video visualization to convey meaningful information contained in video sequences.

CR Categories: I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—Video Analysis; I.3.8 [Computer Graphics]: Applications; I.4.10 [Image Processing and Computer Vision]: Image Representation—Volumetric

Keywords: Video visualization, volume rendering, video surveillance, change detection, image-swept volume.

1 Introduction

In February 2002, it was reported that there were 25 million CCTV cameras in operation worldwide [Wakefield 2002]. In some parts of the world, such as the United Kingdom, it is estimated that on average a citizen is caught on security and traffic cameras 300 times a day. Along with digital camcorders, digitized movies, video conferencing and video emails that are also making their ways into everyday life, it is almost certain that there will be a multi-fold increase in video data in the coming years.

A video is a piece of ordered sequential data, and viewing videos is a time-consuming and resource-consuming process. For example, an increasing problem in the security industry is the ratio of surveillance cameras to security personnel. Imagine that security officers have to review an overnight collection of video tapes when they arrive at their desks in the morning. It is simply not possible for any security officer to study a large number of video tapes everyday. It is hence highly desirable to develop methods for extracting and highlighting interesting features in video sequences.

Automated video processing is a research topic that is of significant importance to the security and entertainment industries. There is a rich collection of techniques for analyzing imagery data, and for computing various statistical indicators. However, there is a general lack of effective techniques to convey complex statistical information intuitively to a layperson such as a security officer, except using line graphs to depict 1D signal levels. Most of the techniques have not reached such an intelligent level that they can be relied upon to make decisions in place of a human.

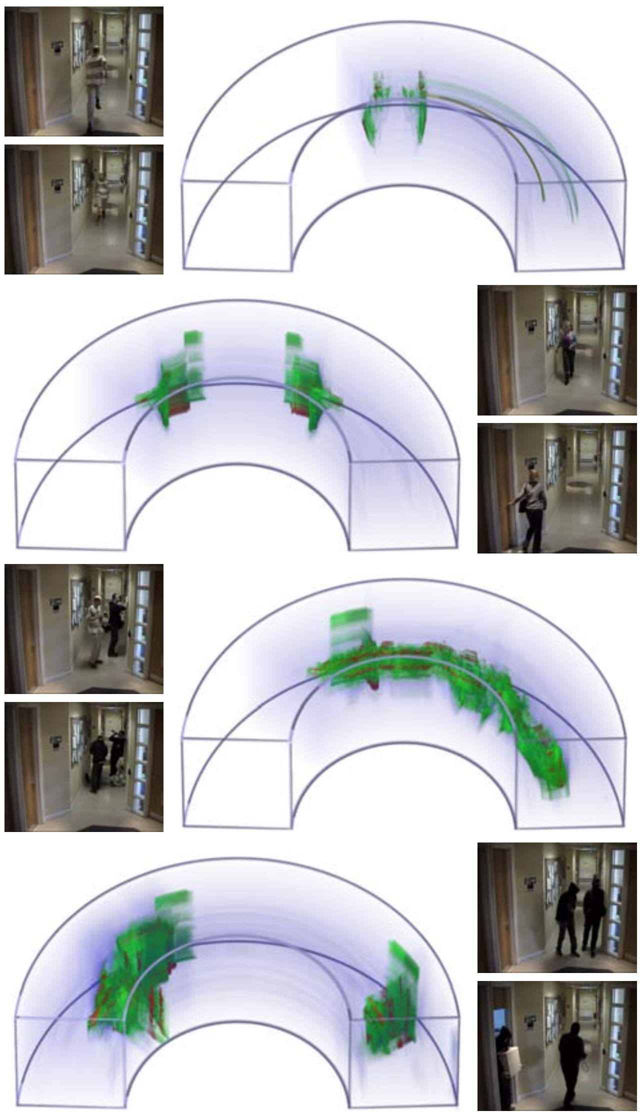


Figure 1: Visualization of four experimental videos based on different scenarios, namely walking, running, mischief and burglary.

[*]e-mail: {csgareth, m.chen}@swansea.ac.uk

In this paper, we present a novel approach to the handling of a very large amount of video data. We propose to employ volume visualization techniques for "summarizing" video sequences, and to render video volumes into appropriate visual representations that can be used to assist in the decision making processes of a human operator. For example, every morning, security officers can be presented with one or a few visualizations for each surveillance camera that has been monitoring a premise during the previous night. Figure 1 shows the visualization of four experimental videos of different staged activities that simulate some typical scenarios which may take place in a university building at night. From the visualizations, one can observe the levels and patterns of the activities recorded. We believe that such visualizations can convey much more information, especially spatial information, than a few statistical indicators or line graphs. With carefully prepared visualizations, the human vision system, perhaps the most intelligent vision system, is able to become accustomed to certain kinds of "normal" visual patterns, and react to unusual levels or patterns of activities that need further investigation. Video visualization can also be used to assist in processing videos, such as video segmentation, and video annotation.

Video data is a type of volume data. Hence the key to our approach is the volume visualization technology, which has been successfully and extensively deployed in medical imaging and scientific visualization. Localized statistical indicators of video data can also be represented in a volumetric form. This conceptual similarity allows us to utilize some powerful volume visualization and volume graphics techniques, such as, *volumetric scene graphs*, *opacity and color transfer functions* and *spatial transfer functions*.

In Section 2, we will briefly review the previous work on video processing, focusing on methods for change detection, and recent work on video cube rendering. In Section 3, we will propose three hypotheses and outline our contributions in this context. In Section 4, we will consider a conceptual pipeline for capturing, managing, processing, rendering and visualizing videos. We will briefly describe the design and development of a prototype system, called V^3 (short for Volume Visualization for Videos). In Section 5, we will discuss various ways for extracting and creating feature volumes, and we will focus on several image comparison metrics for detecting changes in video sequences. This is followed by Section 6, where we will describe the use of a set of volume modeling and rendering techniques for video visualization. In Section 7, we will present some visualization results and discuss the effectiveness of different change detection methods for presenting visual features in video visualizations. We will offer our concluding remarks and an indication of future work in Section 8.

2 Related Work

In order to overcome the sequential and time-consuming process of viewing video [Yeo and Yeung 1997], a noticeable amount of effort has been made, largely by the image processing and vision community, to devise methods for processing video data automatically. One of the research focuses is *change detection*. A common goal in change detection is to ascertain image differences that relate to object changes in a scene. [Narasimhan et al. 2002] outlined many factors, such as reflectance, illumination and atmospheric conditions, which might complicate such a process. A variety of methods were proposed, including thresholding [Rosin 1997], different image comparison metrics [Young et al. 1999; Zhou et al. 2002], morphological filters [Stringa 2000], statistical models [Brocke 2002], combined audio-visual analysis [Tsekeridou et al. 2001], linear dependence models [Durucan and Ebrahimi 2001a; Durucan and Ebrahimi 2001b], color edge detection [Cavallaro and Ebrahimi 2001], and homomorphic filtering [Toth et al. 2000]. Many researchers studied video processing in the context of video surveillance [Collins et al. 2000], for instance, monitoring crowds [Yin et al. 1996], monitoring vehicles [Cutler et al. 1999], and recognizing pedestrians [Vannoorenberghe et al. 1997].

However, two problems remain in automatic video processing: (i) How will the results of video processing, such as detected changes, be communicated to human operators? Statistical results are not easily comprehensible, while sequences of difference images again require sequential viewing. (ii) How reliable will these automatic techniques be in different circumstances? It is generally difficult to develop an automatic video processing technique that can adapt to different situations with little or no calibration.

An alternative approach is to help users obtain an overview of a video by extracting interesting information from video data, and present the information to users in a meaningful way. One suggestion was to use browsing techniques for viewing a video like flipping through a book [Yeo and Yeung 1997]. A number of researchers noticed the structural similarity between video data and volume data commonly seen in medical imaging and scientific computation. For the latter, there is a large collection of methods [Lorensen and Cline 1987; Levoy 1988] that enable 3D information in a volumetric dataset to be selectively rendered into a single 2D image. Attempts were made to render videos as inter-related image frames by [Hertzmann and Perlin 2000], and as volume data sets by [Klein et al. 2002]. Both focused on non-photorealistic rendering for creating an artistic visual experience, and both treated videos as *solid video cubes or cuboids*. There was also a recent report on an initial investigation, in the context of sign language analysis, into the volumetric representation of object movement in a simple video sequence [Hajek 2002].

3 Our Hypotheses and Contributions

Considering video visualization as a new scientific subject, we propose the following three hypotheses:

1. Video visualization is an (i) intuitive and (ii) cost-effective means of processing large volumes of video data.

2. Well constructed visualizations of a video are able to show information that numerical and statistical indicators (and their conventional diagrammatic illustrations) cannot.

3. Users can be accustomed to visual features depicted in video visualizations, or can be trained to recognize specific features.

To ascertain these hypotheses will no doubt require a substantial amount of scientific investigation and technical development. As no previous work on this subject was found in the literature, we hope this paper will serve as a brave path-finder. Our main contributions include:

- We have initiated an original investigation into the subject, and offered a general solution by utilizing volume visualization techniques, focusing on difference volumes, spatial and opacity transfer functions, and stream-based rendering.

- We have designed and implemented a video visualization pipeline by integrating a collection of techniques, and we have demonstrated that it is technically feasible to offer video visualization as a practical tool to applications such as video surveillance and video segmentation.

- We have conducted several case studies, including television programmes, and indoor and outdoor video sequences. The results of these studies have provided the first set of evidences to support hypotheses (1) and (2).

For the third hypothesis, we believe that some comprehensive user studies are necessary, but we are confident about the likelihood of a positive conclusion.

4 Video Visualization Pipeline

A video visualization pipeline is a data flow pipeline, consisting of a series of functional components, namely *video capture* $\Rightarrow$ *data communication* $\Rightarrow$ *data management* $\Rightarrow$ *video processing* $\Rightarrow$ *video visualization*. Each component may accommodate a range of mechanisms and techniques. For example, the video capture component may capture videos from a set of security cameras, or be a tool that receives video emails. The video processing component may house a collection of image processing techniques and statistical methods.

V^3 – *Volume Visualization for Videos* – is a prototype system designed to demonstrate the technical feasibility of such a pipeline. Its primary design objective is to facilitate quick analysis of recently archived video data, such as in the security industry, through the use of volume visualization. This objective is reflected strongly in the design of the V^3 system architecture and user interface (Figure 2).

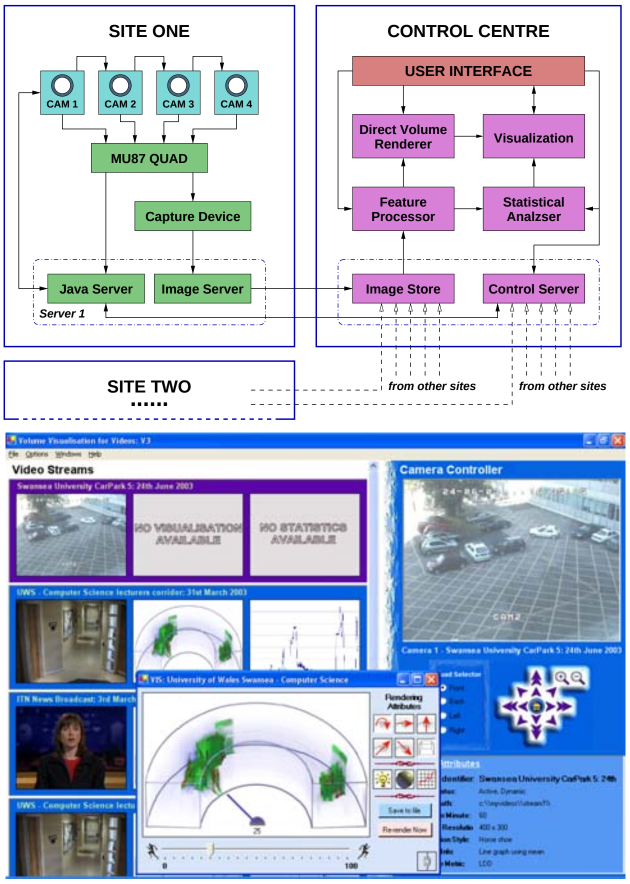


Figure 2: The system architecture and UI of V^3.

V^3 allows multiple sites to be monitored concurrently in real-time from a single control center. At each remote site, we have a set of cameras that can be interactively controlled (e.g., EVI-D31/B Sony video cameras). The imagery data captured by the cameras are combined by, and transported through, an MV87 Quad box, with which an individual or combined view can be selected at the control center. The main software system of V^3 is expected to be installed in a control center, where users can interactively control the remote cameras, select views, setting up recording processes, and most importantly "visualizing" the captured data in many forms.

There are three major algorithmic modules in the software framework of V^3, namely *feature processor*, *statistical analyzer* and *volume renderer*. The feature processor module consists of a number of image processing filters and image comparison metrics. The module takes the raw imagery data as inputs, and generates appropriate outputs for the statistical analyzer and volume renderer modules. The statistical analyzer takes inputs from the feature processor module and produces numerical statistical indicators, which are then forwarded to the visualization module where the statistical indicators are presented as 2D charts (such as line graphs). The volume renderer module handles only volumetric data, which includes the raw video data as well as that generated by the feature processor module. This modular design gives us the flexibility to replace existing metrics, filters and algorithms, and add new ones, whenever necessary.

One of the design objectives of V^3 is to provide novice users with some intuitive but powerful visual representations that facilitate a quick decision-making process. After having experimented with many visual designs, we selected five representations as standard options in V^3 for the default display as shown in Figure 3.

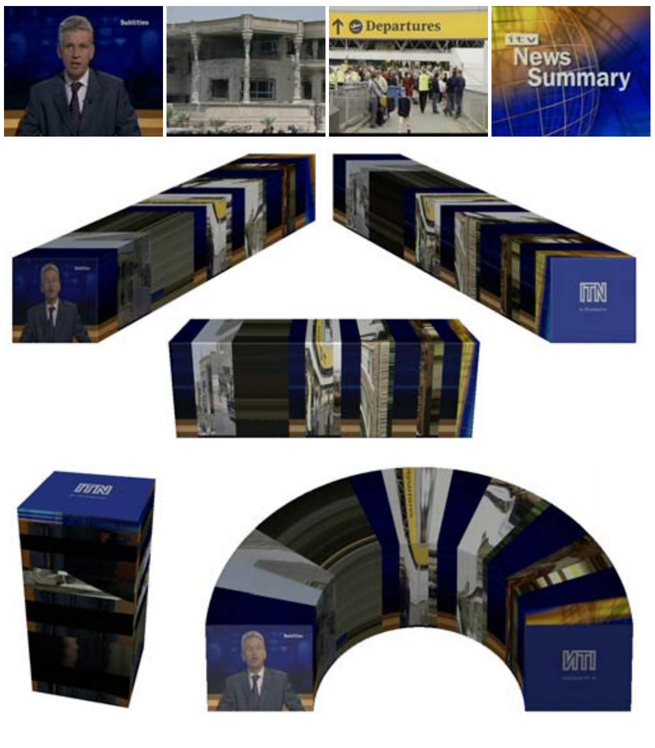


Figure 3: A video is a volumetric object. V^3 provides five standard visual representations of a video object.

The Microsoft Visual C# .NET development environment has been used to implement the main software components of V^3, though many filters, metrics and algorithms were first implemented in C and tested in a Linux environment.

5 Creating Feature Volumes

A video data set V is composed of a series of images $I_1, I_2, \ldots, I_{tn}$, where all images are normally of the same resolution $xn \times yn$. Hence V can be considered as a collection of *voxels* that are organized into a 3D regular grid as:

$$V = \{v_{x,y,t} | 1 \le x \le xn, 1 \le y \le yn, 1 \le t \le tn\}.$$

Each voxel v is addressed by its grid coordinates (x, y, t), and is associated with one or more scalar values representing imagery properties such as intensity and color components. In *volume visualization*, such a structure is commonly referred to as a *volume data set*, *3D raster* or a *volume buffer*. Because the temporal dimension t is of a different nature from that of the spatial dimensions x and y, V should normally be manipulated as an anisotropic grid, whenever the spacing between neighboring voxels is a matter of interest.

The primary objective of video visualization is to extract *meaningful* information from original video data sets, i.e., solid video cuboids. In principle, the extracted information can be represented using any data type. In the context of this work, we concentrate our discussions on information that is suited to volumetric representations. We conveniently call a volumetric representation of such extracted information a *feature volume*.

Given an RGB video volume V, a feature volume F can be constructed using a filter as $F = \Theta(V)$. In general, such a filter may operate on a sub-domain of V, and generate a sub-domain of F. Some example filters considered in this work include:

- *color space converters* – Each filter converts a video volume in the RGB space to that in an alternative color space such as YIQ and HSV.
- *opacity filters* – Each filter creates an opacity feature volume that highlights or de-highlights particular components of an RGB volume. Figure 4 shows a visualization supported by such a filter that determines the opacity of each voxel based on the hue values and edge properties in a small window associated with that voxel.
- *change detection filters* – Each filter typically creates a feature volume that represents the magnitude of temporal changes in a video volume V. This is the focus of the following discussions in the rest of this section.

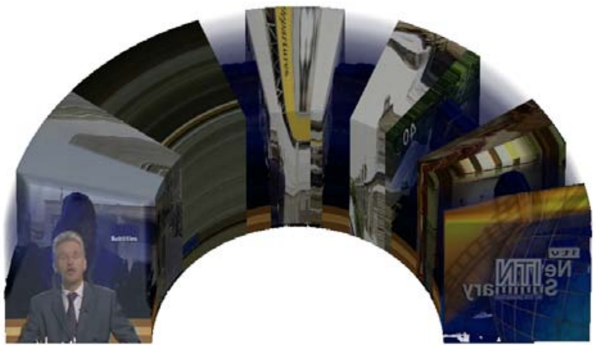


Figure 4: The application of an opacity filter.

We have considered two types of temporal changes, namely *relative difference* between consecutive images in V:

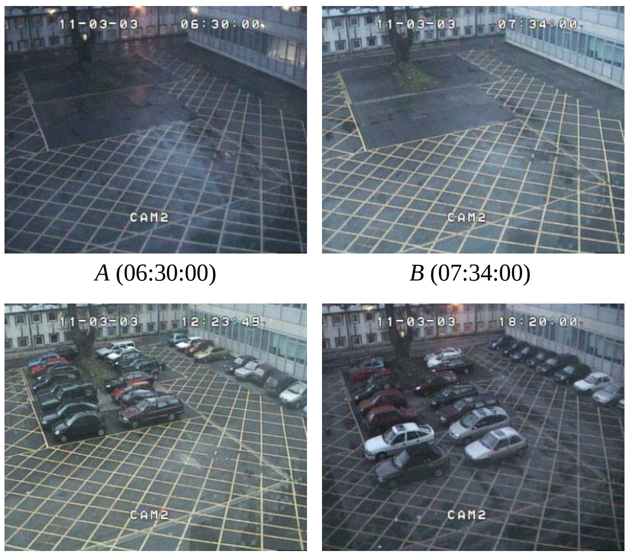
$$F_r = \{\Delta(I_1, I_2), \Delta(I_2, I_3), \ldots, \Delta(I_{tn-1}, I_{tn})\}$$

and *absolute difference* between a reference frame R and each frame in V:

$$F_a = \{\Delta(I_1, R), \Delta(I_2, R), \ldots, \Delta(I_{tn}, R)\}$$

where F_r and F_a are the corresponding feature volumes, and Δ is an image comparison metric that returns a grey-scale image representing the magnitude of changes between two input images. F_r typically gives an indication of the pace of movement or changes, and for instance, it can be used to depict the boundaries between different segments in a TV news programme and the speed of cars on a road. F_a usually gives a good indication of the scale of occupancy, such as the size of a pedestrian crowd in a shopping center. We will further examine the use of these two types of feature volumes, with some examples, in Section 7.

There is a large collection of image comparison metrics in the literature [Young et al. 1999; Zhou et al. 2002]. We have studied the effectiveness of several image comparison metrics in the context of a number of case studies, three of which are discussed in this paper. They are (i) a short TV news programme of 1262 frames recorded from a terrestrial broadcast (Figure 3), (ii) a set of indoor experimental videos, each of 90 seconds and 1770 frames (Figure 1), and (iii) a 12-hour outdoor surveillance video of a university car park of 662 frames (Figure 5). All frames are of the same resolution, i.e., 352×288.



A (06:30:00) *B* (07:34:00)

C (12:23:49) *D* (18:20:00)

Figure 5: Four images extracted from an outdoor surveillance video of a car park. Image B will be used as a reference image in computing absolute differences in Figure 6.

All our image comparison metrics have a built-in RGB-to-YIQ converter. They take RGB images as the inputs, but computes difference in the YIQ space, often using only the luminance channel Y. In order to differentiate the effectiveness of video visualization in *depicting features to users*, and that of computer vision in *identifying features for users*, it is important to consider some less sophisticated, or algorithmically trivial metrics. Among many metrics studied, four have been found interesting. They are:

- Y-DIF(I_1, I_2) – *simple difference metric* – It takes two input images, I_1 and I_2, and computes a grey-scale output image O where each pixel represents the linear distance between the Y-values of two corresponding pixels in I_1 and I_2 respectively.
- Y-NMSE(I_1, I_2) – *normalized mean squared error metric* – Instead of the linear distance, it computes the squared distance (i.e., error) between the Y values of each pair of corresponding pixels. The name of the metric is inherited from the corresponding statistical indicator that calculates the mean of the squared errors of all pairs of pixels in two images. In addition, the Y-component of each input image is normalized based on its mean value and standard deviation prior to the computation of mean squared errors. This may reduce the luminous difference caused by different lighting and atmospheric conditions.
- IQ-DIF(I_1, I_2) – *color difference metric* – It computes the angle between the IQ vectors of the two corresponding pixels in I_1 and I_2, and sets the corresponding pixel value in O to the angle. It gives a result similar to that obtained by computing the hue difference in the HSV space.

- Y-LDD(I_1, I_2) – *linear dependence detector (LDD)* – We use the improved version proposed by [Durucan and Ebrahimi 2001b]. This is an illumination invariant change detector, and it examines the changes from a reference image $R = I_2$ to another image I_1 using a small window. Given a $k \times k$ window centered at (x, y), for each of the two input images, we place the Y values of all pixels in the window into a vector, that is, vector $M = (m_1, m_2, \dots)$ for image I_1, and $N = (n_1, n_2, \dots)$ for R. We normally increase the values in M and N by one to ensure no zero component is in either vector. Based on the algebraic properties of the two vectors, the level of dependence between the two vectors can be measured by:

$$c = \left(\frac{1}{k^2} \sum_{i=1}^{k^2} \frac{m_i}{n_i} \right)^2 - \frac{1}{k^2} \sum_{i=1}^{k^2} \frac{m_i}{n_i}$$

  When $c = 0$, Y-LDD detects no changes; when $c < 0$, it detects illumination changes; when $c > 0$, it detects other changes, including object changes.

Figure 6 shows the application of the above four metrics when comparing images in Figure 5. As image B represents an empty car park in a reasonably good lighting condition, it is chosen as a reference image. Images *A*, *C* and *D* were chosen as they exhibit different levels of activities and different lighting conditions. The value ranges of all resultant images have been re-mapped to the [0, 255] domain for maintaining a fair comparison in the evaluation. All resultant images have been inverted for clearer printing. Hence, in Figure 6, the darker the color is, the higher the level of changes detected.

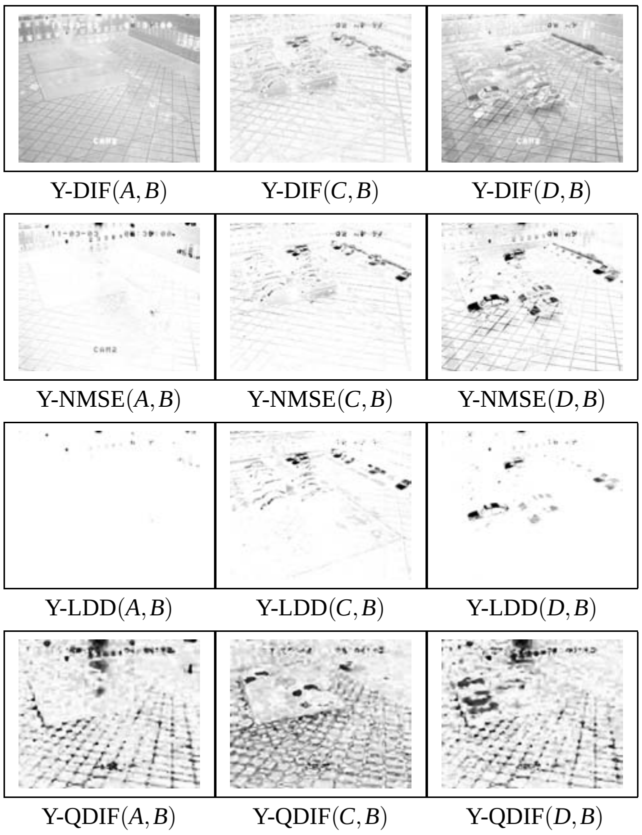


Figure 6: Images A, C and D in Figure 5 are compared with the reference image, B, using four different metrics.

From all three figures, we can see that IQ-DIF does not perform as well as what one would expect. This is partially due to the fact that all images were JPEG-compressed by the image capturing device. The compression seems to be optimized for luminance at the cost of redistributing colors within small regions across the image. IQ-DIF is also not so effective when handling noise in a video sequence. It performed poorly when it was applied to the TV news programme recorded from terrestrial broadcast.

Y-DIF seems to be affected badly by the lighting conditions, and has a lot of difficulties in distinguishing object changes from luminance changes. Y-NMSE has substantially reduced the level of false detection of changes, but it suffers from the fact that global normalization is influenced by dynamic changes such as the movement of cars and office lights, in the scene. In other words, it is not able to maintain a consistent correction of lighting conditions. For Y-LDD, we used a 3×3 window for the above tests. It has shown its effectiveness in illumination-invariant change detection. The criss-cross yellow lines painted on the ground is hardly detected in all three resultant images of Y-LDD. The pixel values of the images returned by Y-NMSE and Y-LDD are normally very low. Appropriate re-scaling is thus necessary prior to the generation of output images. In practice, it is essential to apply a constant scaling factor for an entire video sequence.

6 Rendering Video/Feature Volumes

A video volume and its associated feature volume(s) can be treated exactly as conventional volume data sets during rendering. From such data sets, we can construct a *spatial object* **o**, which is composed of a set of geometrically-bounded attribute fields $(A_0, A_1, \dots, A_k)$ [Chen and Tucker 2000]. Let $\mathbb{R}$ denote the set of all real numbers, and $\mathbb{E}^3$ denote 3D Euclidean space. Each attribute field is a scalar field function $A : \mathbb{E}^3 \to \mathbb{R}$. A typical raw RGB video data set is thus a discrete specification of a spatial object with three attribute fields, namely red, green and blue channels. Its bounding box in effect defines a solid video cuboid, and its volume contents define a solid texture. In V^3, each spatial object **o** is defined with four attribute fields, (O, R, G, B), namely opacity, red, green and blue. For example, the spatial object **o** shown in Figure 3 is in fact associated with a uniform, fully opaque, opacity field within the bounding volume.

Opacity and color transfer functions (which are often referred to simply as transfer functions) are an intrinsic part of volume visualization. We may use color transfer functions to indicate different magnitudes of changes as in Figure 1, where red depicts a higher level of change detected and green depicts a lower level. In this case the color fields of each spatial object is defined upon a feature volume. We may use opacity transfer functions to remove or de-highlight parts of a spatial object. As shown in Figure 4, we can assign a feature volume to the opacity field of a spatial object, which turns the parts with blue as the dominant wavelength into translucent amorphous matter. In the visualization, it is noticeable that the blue background behind the newscaster has almost disappeared from the video volume.

A technique called *image-swept volume* [Winter and Chen 2002] has been employed to facilitate the horseshoe view, which in general conveys more information than the other four views if they are constrained by the same display space. Instead of deforming a video volume during modeling, we associate the object with a *spatial transfer function*, $\Psi : \mathbb{E}^3 \to \mathbb{E}^3$. Ψ defines a sweeping transformation for every point p in $\mathbb{E}^3$. Ψ is used to modify the sampling position of a scalar field A during rendering in the form of $A'(p) = A(\Psi(p))$. Rendering of a spatial object using a ray casting algorithm is essentially a discrete sampling process for evaluating scalar fields. With Ψ, an evaluation of A' at p implies the evaluation of A at $q = \Psi(p)$.

The spatial transfer function for the horseshoe view is a semi-circular sweep. Consider our video volume is defined in a normalized coordinate system of a domain $[0,1]^3$. Let $r = \big((p_x - 0.5)^2 + (p_y - 0.5)^2 + (p_z - 0.5)^2\big)^{1/2}$ and $\phi = \arctan(p_z - 0.5, p_x - 0.5) \in [-\pi, \pi]$. We have:

$$q_x = \begin{cases} 2-4r & r \in [0.25, 0.5] \\ 0 & r \notin [0.25, 0.5] \end{cases} \qquad q_z = \begin{cases} 1-\phi \div \pi & \phi \geq 0 \\ 0 & \phi < 0 \end{cases}$$

$$q_y = 1 - p_y$$

[Chen et al. 2003] recently demonstrated that spatial transfer functions can be defined as spatial objects, and they can be integrated into a volume scene graph [Chen and Tucker 2000] in the same way as conventional spatial objects. As V^3 is a special purpose visualization system, we purposely moderate its complexity and memory consumption by not equipping it with the facilities for building an arbitrary scene graph. Instead, we have a reconfigurable object tree with a fixed set of built-in tree nodes. For the five standard visual representations shown in Figure 3, we have (a) an internal node with a union operator, (b) an internal node with a spatial transfer operator, (c) a leaf node for a video object, which a video volume and/or a feature volume may be loaded onto, (d) a leaf node for a box frame, and (e) a leaf node for an object that defines a sweeping function using a set of scalar fields. Depending on the view selection and the requirement of the box frame, the system reorganize the object tree into one of the four optional configurations (Figure 7).

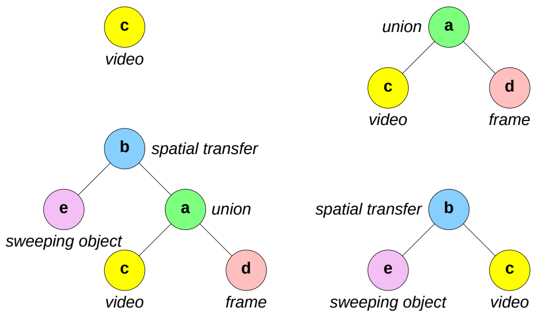


Figure 7: Four configurations of object trees in V^3.

A *stream-based rendering algorithm* can benefit video visualization enormously. Video data sets and their feature volumes can be excessively large, containing thousands of frames. Such data sets not only consume a huge among of memory, but also put a lot of strain on the rendering speed largely due to memory swapping. However, in many applications, such as surveillance, videos are coming in streams as image frames. It is hence desirable to start the rendering process as soon as the first frame arrives, and to continue the construction of a visualization progressively following the receipt of each new frame. This strategy is well suited for constructing summary visualizations from overnight video recordings.

With the ray casting method, we need to build both front-to-back and back-to-front rendering into the stream-based algorithm, because of the order of arriving video frames. Consider the five visual representations in Figure 3. The vertical view and the downward diagonal view require back-to-front ray casting, whilst the upward diagonal view requires front-to-back ray casting. The horseshoe view requires both mechanisms, with front-to-back on the left and back-to-front on the right. Similarly the horizontal view also requires both mechanisms.

In general, for an arbitrary view, one may determine the ray direction for each pixel in an initialization stage where the ray-box intersection is computed. One must note that it is possible for a ray to enter a horseshoe object twice, if we allow arbitrary viewing positions. Fortunately, the ray casting direction will always remain the same for both intersections. However, this observation cannot be generalized to arbitrary spatial transfer functions.

For the standard viewing options in V^3, the ray casting direction for each pixel, and the ray-box intersection points, are predetermined and stored in a set of system files. The data is loaded into a “t-buffer” before rendering starts. The t-buffer also maintains the current rendering status, including the accumulated opacity and color, and the point on the ray where the last casting paused. The algorithm is outlined below:

```
initialize TBuffer;
NumRenderedPixels := 0;
while NumRenderedPixels < TotalPxels do
    wait for new video frames;
    for each pixel P do
        End := RayCast(TBuffer[P]);
        if End = TRUE then
            NumRenderedPixels := NumRenderedPixels + 1;
        end-if;
    end-for
    display intermediate visualization;
end-while;
for each pixel P do
    add background color;
end-for;
display final visualization;
```

With this stream-based algorithm, the system needs only to keep a small collection of “active” frames of an incoming video stream and its feature stream(s) in an in-core volume buffer as shown in Figure 8. The rendering of the each intermediate visualization is on average less than 0.2 second on a 2 GHz Pentium station.

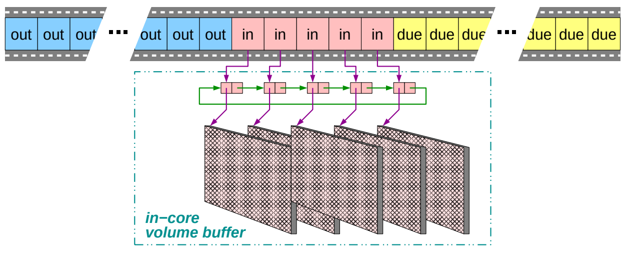


Figure 8: Dynamic management of the in-core volume buffer.

7 Results and Remarks

Referring to the three case studies mentioned in Section 5, we have examined the effectiveness of the image comparison metrics in conjunction with video visualization. Here we report a selection of our tests and findings.

One common task in video segmentation is to identify transition frames in a video and select a representative image for each segment. Any automatic algorithm for the task will face various challenges such as blending between segments, fast camera movement and flash photography. It is most likely that its parameters would also require frequent adjustments for different scenes. Hence errors are inevitable. For example, we applied Y-DIF, Y-NMSE and Y-LDD metrics to the TV news video in Figure 3 to compute F_r, the relative difference between consecutive images in the video volume. Figure 9 shows a line graph for each metric depicting the mean intensity of every frame in F_r. All three metrics have resulted in some errors, in comparison with the transition frames determined manually. Verifying the correctness of such a process usually requires the viewing of the video sequentially, which nevertheless undermines the advantages of using an automatic algorithm.

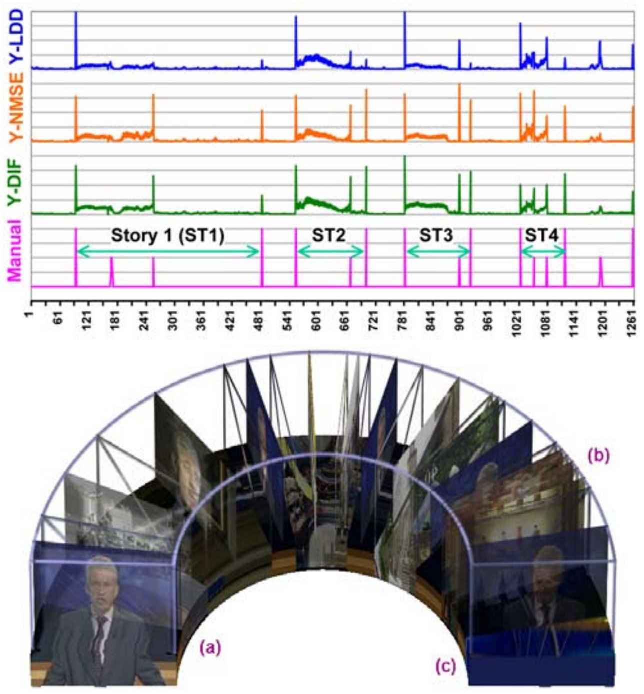


Figure 9: Line graphs depicting the change-detection results produced by Y-LDD, Y-NMSE, Y-DIF, and human decision. A visualization (based on Y-NMSE) of transition frames and representative images can help identify errors in the numerical results.

Video visualization can provide an effective means to the verification, for instance, using the 3D visualization in Figure 9, which shows the transition frames (in the form of ⊠) identified using Y-NMSE and representative images in individual segments. The bottom quarter of the video is also displayed to offer a visual cue for different segments. By examining the visualization, we can observe some anomalies that indicate possible errors. These include (a) the sharing of a representative image by two segments, (b) a possibly over-segmented region featuring similar color patterns, and (c) a missing representative frame after the "presenter" segment (cf. the presenter on the left). When future development enables interactive navigation through such a visualization, identifying these errors in video segmentation will become even easier.

In Section 5, we have found that Y-LDD have shown to be capable of detecting object changes in different illumination conditions, and it has generally performed better than other metrics with outdoor images. Y-NMSE was also shown to be reasonably reliable. Figures 10 shows the visualizations of relative and absolute difference volumes of the car park video, which were computed using the Y-LDD and Y-NMSE metrics respectively. All these feature volumes were visualized on their own with a color transfer function, indicating the scale of changes (i.e., red for large intensity changes, green for medium and blue for small).

The visual representations of the relative difference indicates the level of activities during the recording period, that is, movement of cars. The similar pattern of activities are shown in the visualizations created with both Y-LDD and Y-NMSE. In general a feature volume of relative difference created by Y-LDD may contain more noise than that by Y-NMSE as Y-NMSE seems to be more effective in

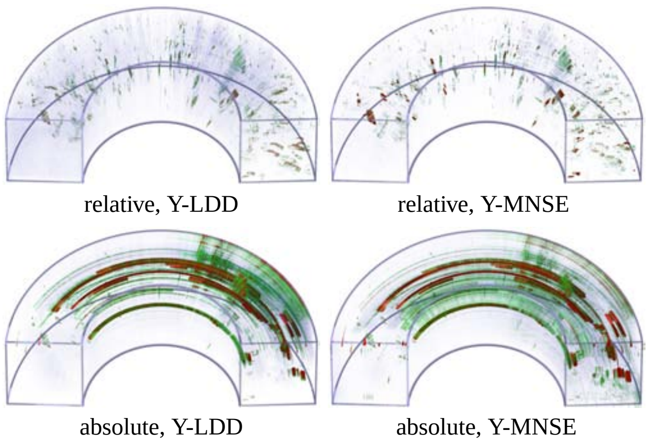


Figure 10: The visualization of relative and absolute difference volumes computed from the car park video sequence.

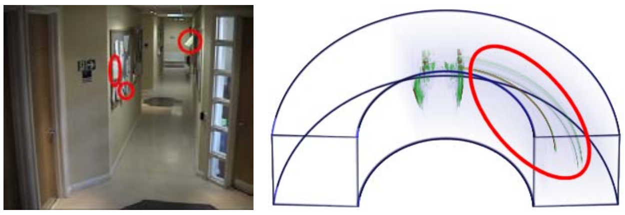
dealing with images taken in similar lighting conditions.

The feature volumes showing absolute difference have resulted in interesting visualizations, where the swept lines indicate many stationary cars in most parts of the recording period. Such a visualization shows the level of usage of the car park, with little occupancy in the early morning, a full car park during the day, and some dynamic activities in the evening when staff were leaving for home and evening students were coming to the university. The Y-NMSE resulted in a slightly more noisy visualization, as it cannot remove unwanted features such as the criss-cross yellow lines as effectively as Y-LDD. On the video, there was a major change of the weather condition during the afternoon, and this change is clearly visible from both visualizations A line graph depicting mean intensity of each frame could easily misinform us of some extra activities or occupancy. However, in the visualizations, it is much easier to discard such changes as the amorphous green patterns are perpendicular to the time line.

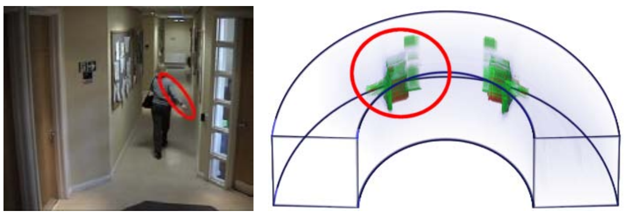
We have found that Y-LDD and Y-NMSE did not perform as effectively as Y-DIF for the indoor experimental videos. All visualizations in Figure 1 were generated with absolute difference detected by Y-DIF. Some interesting features can be identified from such visualizations, once an observer is accustomed to some regular patterns. For example, as shown in Figure 11(a), after a man ran up the corridor, some changes to papers on the noticeboards remained, and such changes were shown as sweeping lines in the corresponding visualization (similar to parked cars). A wavy pattern in Figure 11(b) was largely the result of the arm movement of the lady walking up the corridor. The opening of an office door also has a noticeable pattern as shown in Figure 11(c). In fact, one can find a few similar patterns in Figure 1, all resulting from the opening of the door on the left of the scene.

8 Conclusions

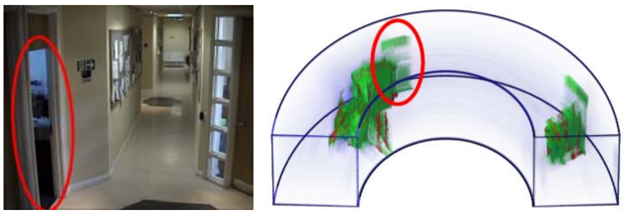
We have described an approach that can effectively "summarize" a video sequence and can be deployed to deal with the problem of rapid explosion of video data. We have shown that video data can be processed and visualized in the same manner as other volumetric data. We have examined several image comparison metrics, and have found that in different circumstances, some perform better than others. In well-controlled environments, such as the ITN news video and the indoor experimental videos, simple Y-DIF can be used effectively to create feature volumes. In more dynamic conditions, Y-LDD and Y-NMSE performed better. With the aid of



(a) changes that remain for a period.



(b) walking with moving arms



(c) door opening

Figure 11: Visual patterns in the visualizations for the experimental videos, where corresponding features are manually highlighted.

three sets of examples, we have provided the first set of evidences to support two of the three hypotheses outlined in Section 3, and have demonstrated the usefulness of video visualization in applications such as video surveillance, and video processing and labeling.

Our future work will be focused on further investigation into image comparison metrics in order to improve the effectiveness of change detections in outdoor conditions, and the capabilities of depicting more features.

Acknowledgments

The authors are particularly grateful to Justin Biddle, David Clark, Emma Jeffery and Mark Kiddell for their assistance in making some experimental videos. They also wish to acknowledge various financial supports, in the form of studentships and equipment grants, received from UK-EPSRC, University of Wales Swansea, DHI Powell Memorial Fund, and Knowledge Exploitation Fund.

References

BROCKE, M. 2002. Statistical image sequence processing for temporal change detection. In *Proc. 24th DAGM Symposium: Pattern Recognition*, Springer, LNCS 2449, Zurich, Switzerland, 215–223.

CAVALLARO, A., AND EBRAHIMI, T. 2001. Change detection based on color edges. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS-2001)*, Sydney, Australia, 141–144.

CHEN, M., AND TUCKER, J. 2000. Constructive volume geometry. *Computer Graphics Forum 19*, 4, 281–293.

CHEN, M., SILVER, D., WINTER, A. S., SINGH, V., AND CORNEA, N. 2003. Spatial transfer functions – a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics 2003*, Tokyo, Japan, Eurographics/ACM Publications, 35–44.

COLLINS, R., LIPTON, A., KANADE, T., FUJIYOSHI, H., DUGGINS, D., TSIN, Y., TOLLIVER, D., ENOMOTO, N., AND HASEGAWA, O. 2000. A system for video surveillance and monitoring. Tech. Rep. CMU-RI-TR-00-12, Carnegie Mellon University, Robotics Institute, May.

CUTLER, R., SHEKHAR, C., BURNS, B., CHELLAPPA, R., BOLLES, R., AND DAVIS, L. 1999. Monitoring human and vehicle activities using airborne video. In *Proc. 28th Applied Imagery Pattern Recognition Workshop (AIPR)*, Washington, DC.

DURUCAN, E., AND EBRAHIMI, T. 2001. Change detection and background extraction by linear algebra. *Proceedings of the IEEE 89*, 10, 1368–1381.

DURUCAN, E., AND EBRAHIMI, T. 2001. Improved linear dependence and vector model for illumination invariant change detection. In *Proc. SPIE*, vol. 4303, San Jose, California.

HAJEK, J. 2002. Time reconstruction of video sequence. In *Electronic Proc. Central European Seminar on Computer Graphics*, http://www.cg.tuwien.ac.at/studentwork/CESCG/CESCG-2002/.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *Proc. 1st International Symposium on Non-Photorealistic Animation and Rendering*, Annecy, France, 7–12.

KLEIN, A. W., SLOAN, P. J., FINKELSTEIN, A., AND COHEN, M. F. 2002. Stylized video cubes. In *Proc. ACM SIGGRAPH Symposium on Computer Animation*, San Antonio, Texas, 15–22.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3, 29–37.

LORENSEN, W., AND CLINE, H. 1987. Marching cubes: a high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH'87*, vol. 21(4), Anaheim, California, 163–169.

NARASIMHAN, S., WANG, C., AND NAYAR, S. 2002. All the images of an outdoor scene. In *Computer Vision – ECCV (3)*, Springer, LNCS 2352, Copenhagen, Denmark, 148–162.

ROSIN, P. 1997. Thresholding for change detection. In *Proc. 8th British Machine Vision Conference*, Essex, UK, 212–221.

STRINGA, E. 2000. Morphological change detection algorithms for surveillance applications. In *Proc. 11th British Machine Vision Conference*, Bristol, UK, 402–411.

TOTH, D., AACH, T., AND METZLER, V. 2000. Illumination-invariant change detection. In *Proc. 4th IEEE Southwest Symposium on Image Analysis and Interpretation*, Austin, Texas, 3–7.

TSEKERIDOU, S., KRINIDIS, S., AND PITAS, I. 2001. Scene change detection based on audio-visual analysis and interaction. In *Multi-Image Analysis*, Springer, LNCS 2032, Dagstuhl, Germany, 214–225.

VANNOORENBERGHE, P., MOTAMED, C., BLOSSEVILLE, J.-M., AND POSTAIRE, J.-G. 1997. Automatic pedestrian recognition using real-time motion analysis. In *Proc. ICIAP 97 Image Analysis and Processing*, LNCS 1311, Florence, Italy, 493–500.

WAKEFIELD, J. 2002. Watching your every move. *BBC News Online* (7 February). http://news.bbc.co.uk.

WINTER, A., AND CHEN, M. 2002. Image-swept volumes. *Computer Graphics Forum 21*, 3, 441–450.

YEO, B.-L., AND YEUNG, M. 1997. Retrieving and visualizing video. *Communications of the ACM 40*, 12, 43–52.

YIN, J. H., ZHANG, X., VELASTIN, S. A., AND DAVIES, A. C. 1996. Incident detection in pedestrian traffic using image processing. In *Proc. 8th IEE International Conference on Road Traffic Monitoring and Control*, London, UK, 115–119.

YOUNG, S., FORSHAW, M., AND HODGETTS, M. 1999. Image comparison methods for perimeter surveillance. In *Proc. 7th IEE International Conference on Image Processing and its Applications*, 465, 799–803.

ZHOU, H., CHEN, M., AND WEBSTER, M. F. 2002. Comparative evaluation of visualization and experimental results using image comparison metrics. In *Proc. IEEE Visualization 2002*, Boston, MA, 315–322.